METHOD FOR TESTING IEEE 1394 CONTROLLERS

This application incorporates by reference Taiwanese application Serial No. 090102690, Filed February 7, 2001.

BACKGROUND OF THE INVENTION

1.Field of the Invention

[0001] The invention relates in general to a method for testing 1394 controllers, and more particularly to a method for testing 1394 controllers in a multiple system.

2.Description of the Related Art

[0002] Within the field of computer application nowadays, information is commonly shared or transmitted between two apparatus, such like two hosts, a host and a peripheral, or two peripherals. The larger the file size is, the more time for transmission is required. It takes a long time to transmit a file with large size by a lower transmission standard and brings users inconvenience to use.

[0003] It is therefore urgent to require a transmission standard supporting high-speed data transmission. As far as the transmission standard in common use is concerned, IEEE 1394 provides an advanced solution for the high-speed transmission and becomes highly valued. The IEEE 1394 standard is also known as iLink, or FireWire.

[0004] After finishing the design of the interface card with a 1394 controller,

the interface card has to be put to the general test and the load flow test to ensure its performance. The test methods presently used are performed under the driver structure in Microsoft's Windows System. The packet of 1394 standard is generated by the test method and received by the interface card to ensure its normalization.

[0005] However, the drawback of the test method is that it is hard to debug during the load flow test. The load flow test is to send out a large amount of packets to the interface card in a very short time for testing the stability of the interface card. The driver under Windows Operating System needs an additional debug program, SoftICE of NuMega company for example, to check the values inside the computer memory and the buffer. It is hard to perform debugging for the test worker since the memory address in the Microsoft's Windows System is a virtual address, which requires a transformation for mapping onto the physical address. It is also hard to trace the wrong packet while problem causing. Besides, it is not easy to monitor the testing status at every moment due to the difficulties of observing the buffer and memory and the inconvenience of changing the buffer setting.

SUMMARY OF THE INVENTION

[0006] It is therefore an object of the invention to provide a method for testing interface cards with IEEE 1394 controllers. The method can be performed in a multiple system and it is no need to be restricted in the Windows Operating System.

[0007] It is another object of the invention to provide a method for testing

interface cards with IEEE 1394 controllers. It can fully control the buffer and is convenient to test in a load flow environment by using this method.

[0008] It is the other object of the invention to provide a method for testing interface cards with IEEE 1394 controllers. The packet of 1394 standard is easily generated by the test method and received by the interface card for automatically checking the test result. Besides, the invention can detect and find out what the problem is by monitoring the buffer during the test. Further, it can easily maintain and add new functions to the test method according to the invention.

[0009] The invention achieves the above-identified objects by providing a method for performing a load flow test among a number of IEEE 1394 controllers. The IEEE 1394 controllers are disposed on a number of interface cards individually while the interface cards are placed in a host or several hosts. The method includes the following steps: The interface cards are first initialized. One of the interface cards is set as a master interface card and the other interface cards are set as slave interface cards. Then, the master interface card initializes a number of communication protocol packets while each slave interface card sends a first ready signal. The master interface card sends the communication protocol packets to the slave interface cards for responding to the first ready signals. The slave interface cards then send a number of second ready signals individually for responding to the communication protocol packets sent by the master interface card. After that, the master interface card starts to perform the load flow test for responding to the second ready signals. After checking the status of every interface cards, the master interface card sends a number of check packets to the slave interface cards; meanwhile,

debugging is performed if any error occurs before sending. The slave interface cards then send a number of third ready signals individually for responding to the check packets. After that, the master interface card sends a number of confirm signals to the slave interface cards for responding to the third ready signals. Finally, the slave interface cards checks the test results for responding to the confirm signals.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] Other objects, features, and advantages of the invention will become apparent from the following detailed description of the preferred but non-limiting embodiments. The description is made with reference to the accompanying drawings in which:

[0011] FIG. 1A-1B illustrates two environments to which the preferred embodiment of the invention is applied;

[0012] FIG. 2A-2D shows a flowchart of the test method according to the preferred embodiment of the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

[0013] This invention is used to put several 1394 controllers to the load flow test. The 1394 controllers are disposed on interface cards individually. The

interface cards can be placed in one or several hosts. Referring to FIG. 1A and 1B, two environments to which the preferred embodiment of the invention is applied are illustrated. As shown in FIG. 1A, the test method according to the invention is applied to a single system. The single system includes a host 10 and interface cards 20 and 30 placed in the host 10. The interface card 20 is connected with the interface card 30 by the cable 25. As shown in FIG. 1B, the test method according to the invention is applied to a multiple system. The multiple system includes two systems. One system consists of a host 40 and an interface card 50. The other system consists of a host 70 and an interface card 60. Besides, the interface card 50 is connected with the interface card 60 by the cable 55. The invention is not restricted to be applied to environments as shown in FIG. 1A and FIG. 1B, it can also be applied to the a multiple environment consisting of several interface cards placed in a host or a multiple environment consisting of several interface cards placed in several hosts.

[0014] The prerequisite of the invention is that all the interface cards to be tested work are in a normal flow condition. The purpose of this invention is providing a method for testing if all the interface cards normally work in a load flow environment.

[0015] The embodiment of the test method is developed in the MS-DOS (Microsoft Disk Operating System) in order to avoid the test method being restricted in the Windows Operating System.

[0016] As shown in FIG. 2A to 2D, the flowchart of the test method according

to the preferred embodiment of the invention is shown. Referring first to Figure 2A, it starts to initialize the interface cards to be tested and allocate memory in step 210. Since there might be several hosts for building the test environment as shown in FIG. 1B, the operating systems of all the hosts are waited for being ready in step 215 to proceed the procedure. In the next step 220, all the interface cards to be tested have to communicate one another for building a test environment. After successfully building the test environment by using all the communication data, one of the interface card is selected as a master interface card and the other interface cards are the slave interface cards. The way to choose the master/ slave interface cards is known by the skilled person in the art and thus omitted without further description. Subsequently, the master interface card and the slave interface cards proceeds to perform different test procedures. As shown in FIG. 2A to 2D, there are dotted line for separating two test procedures. The left one is the test procedure of the master interface card while the right one is the test procedure of the slave interface card. The master interface card and the slave interface card proceed to the master test table and the slave test table respectively and wait the instruction from the test worker as shown in step 225 and 230.

[0017] The master test table includes three choices: (1) initializing communication protocol packets; (2) performing the test; (3) end. It enters step 235 when the test worker chooses to initialize communication protocol packets according to the (1) choice in the master test table. It enters M1 node when the test worker chooses to performing the test according to the (2) choice in the master test table. It enters step 240 and concludes the test procedure of the master interface card when the

test worker chooses the end according to the (3) choice in the master test table.

[0018] In step 235, the communication protocol packets are initialized according to the following three modes: a command mode, edit mode, and random mode. The communication protocol packets are generated by a predetermined command file in the command mode. The communication protocol packets are generated by the key-in content from the test worker in the edit mode. The communication protocol packets are generated at random in the random mode. After initializing the communication protocol packets, it returns to step 225.

[0019] The slave test table includes two choices: (1) performing the test; (2) end. It enters step 245 and concludes the test procedure when the test worker chooses the end according to the (2) choice in the slave test table. It enters S1 node and proceeds the subsequent steps when the test worker chooses to perform the test according to the (1) choice in the slave test table.

[0020] As shown in FIG. 2B, the slave interface card sends a first ready signal to the master interface card in step 250. Later in step 255, the master interface card detects whether or not it receives the first ready signal from the slave interface card in a set period. If the first ready signal is received, it enters to step 257 and the master interface card sends the communication protocol packet to the slave interface card. If not, it returns to the master test table as shown in step 256. This means some errors occur and the test worker has to debug.

[0021] In step 260, the slave interface card detects whether or not it receives

the communication protocol packet in a set period. If the communication protocol packet is received, it enters to step 270 and the slave interface card sends the second ready signal to the master interface card. If not, it returns to the slave test table as shown in step 261. Similarly, this means some errors occur and the test worker has to debug.

[0022] In step 280, the master interface card detects whether or not it receives the second ready signal from the slave interface card in a set period. If the second ready signal is received, it enters to step 285 and the master interface card sends contact signals to all the slave interface card and starts to perform the load flow test. If second ready signal is not received, it returns to the master test table as shown in step 281. This means some errors occur and the test worker has to debug.

[0023] Meanwhile, the slave interface card detects whether or not it receives the contact signal from the master interface card in a set period. If the contact signal is not received, it returns to the slave test table as shown in step 291. This also means some errors occur and the test worker has to debug. If the contact signal is received, it enters to A node.

[0024] Referring to FIG. 2C, the slave interface card starts to perform the load flow test after receiving the contact signal from the master interface card. That is, all the interface card, including the master interface card and the slave interface card, perform the load flow test in step 295. There are a large amount of packets transmitted among interface cards and it waits to complete the command instructed by the packet. For example, the command might be an instruction of reading or writing

the data from the memory of some interface card or a host. In the next step 300, the status of every interface card is checked.

[0025] If there is any error occurred, not only the debugging is performed but also the system needs rebuilding in step 305. After that, in step 306, the master interface card and the slave interface card return to the master test table and the slave test table respectively. As shown in step 310, if there is no error occurred, the master interface card sends check packets to the slave interface cards for the preparation of checking the test results, such as the transmission speed of packets. In step 310, the master interface card proceeds to the M2 node and the slave interface card proceeds to the S2 node.

[0026] Referring to FIG. 2D, in step 315, the slave interface card sends the third ready signal to the master interface card for responding to the check packets. In step 320, the master interface card detects whether or not it receives the third ready signal from the slave interface card in a set period. If the third ready signal is received, it enters to step 325 and the master interface card sends confirm signals to all the slave interface card. If the third ready signal is not received, it returns to the master test table as shown in step 321. This means some errors occur in the testing procedure.

[0027] Meanwhile, in step 330, the slave interface card detects whether or not it receives the confirm signal from the master interface card in a set period. If the confirm signal is not received, it returns to the slave test table as shown in step 331. If the confirm signal is received, the slave interface cards check the results of the load

flow test in step 335, such as the transmission speed of packets.

[0028] In step 336, the master interface card decides whether or not it exits the test. If deciding to exit, the master interface card sends an instruct signal of exiting in step 340 and it returns to the master test table. If deciding not to exit, the master interface card sends instruct signals of resuming to all the slave interface cards in step 345 and then the it returns to the M1 node in FIG. 2B and resumes the test.

[0029] As shown in step 360, the slave interface card detects whether it receives the instruct signal from the master interface card to decide either exiting or resuming. If the instruct signal is not received in a set period, it returns to the slave test table in step 361. If the slave interface card receives the instruct signal of existing, it returns to slave test table. If the slave interface card receives the instruct signal of resuming, it returns to the S1 node of FIG. 2B and resumes the test.

[0030] According to the invention, all the acts of returning to the master/slave test table are monitored. However, it is a normal situation if the act is decided by the test worker, such as step 340 and step 370, and it does mean there is any error occurred. Otherwise, it is an abnormal situation if the act is not decided by the test worker. It means some error occurs and the test worker is able to control and solve the problem by debugging.

[0031] The method for testing the interface cards with controllers as mentioned above provides the following advantages:

[0032] (1) It can fully control the buffer and monitor the memory since the test

method is not performed under the driver structure in Microsoft's Windows Operating System. Besides, it is convenient to test in a load flow environment by using this method.

[0033] (2) The packet of IEEE 1394 standard is easily generated by the test method and received by the interface card for automatically checking the test result.

[0034] (3) It can detect and find out what the problem is by monitoring the buffer and memory during the test.

[0035] (4) The method can be performed in a multiple system and it is no need to be restricted in the Windows Operating System.

[0036] (5) It can easily maintain and add new functions to the test method according to the invention.

[0037] While the invention has been described by way of example and in terms of the preferred embodiment, it is to be understood that the invention is not limited to the disclosed embodiment. To the contrary, it is intended to cover various modifications and similar arrangements and procedures, and the scope of the appended claims therefore should be accorded the broadest interpretation so as to encompass all such modifications and similar arrangements and procedures.